SCHEDULING MASSIVELY PARALLEL MULTIGRID FOR MULTILEVEL MONTE CARLO METHODS

D. DRZISGA*, B. GMEINER†, U. RÜDE†, R. SCHEICHL‡, AND B. WOHLMUTH*

Abstract. The computational complexity of naive, sampling-based uncertainty quantification for 3D partial differential equations is extremely high. Multilevel approaches, such as multilevel Monte Carlo (MLMC), can reduce the complexity significantly when they are combined with a fast multigrid solver, but to exploit them fully in a parallel environment, sophisticated scheduling strategies are needed. We optimize the concurrent execution across the three layers of the MLMC method: parallelization across levels, across samples, and across the spatial grid. In a series of numerical tests, the influence on the overall performance of the "scalability window" of the multigrid solver (i.e., the range of processor numbers over which good parallel efficiency can be maintained) is illustrated. Different homogeneous and heterogeneous scheduling strategies are proposed and discussed. Finally large 3D scaling experiments are carried out including adaptivity.

1. Introduction. Data uncertainties are ubiquitous in many application fields, such as subsurface flow or climate prediction. Inherent uncertainties in input data propagate to uncertainties in quantities of interest. This situation has driven the development of novel uncertainty quantification (UQ) methods; most commonly, using partial differential equations (PDEs) to model the physical processes and stochastic models to incorporate data uncertainties. Simulation outputs are then statistics (mean, moments, cumulative distribution function (CDF)) of the quantities of interest. However, typical sampling-and-averaging techniques for computing statistics quickly become infeasible, when each sample involves the numerical solution of a PDE.

We consider an abstract, possibly nonlinear system of PDEs with uncertain data

$$\mathcal{M}(u;\omega) \;=\; 0 \tag{1}$$

posed on a bounded domain $D \subset \mathbb{R}^d$, where the solution u is sought in some suitable space V of functions $v : D \to \mathbb{R}^k$ with $k \in \mathbb{N}$, subject to suitable boundary conditions. $\mathcal{M}$ is a differential operator depending on a set of random parameters parametrised by an element ω of the abstract sample space $(\Omega, \mathcal{F}, \mathbb{P})$ that encapsulates the uncertainty in the data, with Ω the set of all outcomes, $\mathcal{F}$ the σ-algebra (the "set" of all events), and $\mathbb{P}$ the associated probability measure. As a consequence the solution u itself is a random field, i.e. $u = u(x, \omega)$, with realizations in V.

We are typically only interested in functionals $Q(u) \in \mathbb{R}$ of u. To compute them we need to approximate the solution u numerically, e.g. using finite element methods, which introduces bias error. The cost $\mathcal{C}$ typically grows inverse proportionally to some power of the bias error, i.e. $\mathcal{C} = \mathcal{O}(\varepsilon^{-r})$ where ε denotes the bias error tolerance. This challenging computational task requires cutting-edge parallel computing for two reasons: firstly, real life applications lead to PDE systems that often can only be solved accurately on a parallel computer; secondly, typical uncertainties, such as a random diffusion coefficient $k(x, \omega)$, are spatially varying on many scales.

For low dimensional problems, stochastic Galerkin, stochastic collocation and polynomial chaos methods have been shown to provide efficient and powerful UQ

*Institute for Numerical Mathematics, Technische Universität München, 85748 Garching, Germany (drzisga@ma.tum.de, wohlmuth@ma.tum.de), partly funded by WO671/11-1 (DFG)

†Institute of System Simulation, University Erlangen-Nuremberg, 91058 Erlangen, Germany (bjoern.gmeiner@fau.de, ulrich.ruede@fau.de)

‡Dept. Mathematical Sciences, University of Bath, Bath BA2 7AY, UK (r.scheichl@bath.ac.uk)

tools (see, e.g., [14, 40, 25] and the references therein), but in general their complexity grows exponentially with the stochastic dimension. The cost of sampling methods, such as, e.g., Monte Carlo, does not grow with the stochastic dimension, but classical Monte Carlo is notoriously slow to converge. Multilevel Monte Carlo (MLMC) simulation [15, 6] can reduce significantly the algorithmic complexity by performing as much computational work as possible on coarse meshes. To this end, MLMC uses a hierarchy of discretisations of (1) of increasing accuracy to estimate statistics of $Q(u)$ more efficiently, i.e. using a large number of coarse samples to fully capture the variability, but only a handful of fine samples to eliminate the bias due to the spatial discretisation. Here, we employ multilevel methods not only to accelerate the stochastic part, but also to provide a scalable solver for individual realizations of (1).

Current leading-edge supercomputers provide peak performances of about hundred petaflop/s (i.e. 10^{17} floating point operations per second) [31]. However, they all draw their computational power from parallelism, with processor numbers already at $P_{\max} \approx 10^7$ see [10]. Consequently, designing efficient algorithms for high performance computers is a challenging task today and will be even more so in the future.

MLMC methods are characterized by three algorithmic levels that are potential candidates for parallel execution. As in standard Monte Carlo methods, a sequence of classical deterministic problems (samples) are solved that can be computed in parallel. Additionally one can distinguish between parallelism within an MLMC level and parallelism across MLMC levels. The third algorithmic parallelization level is the solver for the deterministic PDE problem. Indeed, the total number of samples on finer MLMC levels is typically moderate, so that the first two levels of parallelism will not suffice to exploit $P_{\max}$ processors. Parallel solvers for elliptic PDEs are now able to solve systems with 1.1×10^{13} degrees of freedom on petascale machines with compute times of a few minutes [17] when using parallel multigrid methods [5] in an optimized implementation [18].

In this paper, we will illustrate how these different levels of parallelism can be combined and how efficient parallel MLMC strategies can be designed. To achieve this, we extend the massively parallel Hierarchical Hybrid Grids (HHG) framework [3, 19] that exhibits excellent node performance as well as strong and weak scaling behavior [21, 1] to the MLMC setting. Additionally, we use its fast multigrid solver in a novel way to generate spatially correlated samples of the random diffusion coefficient. In the MLMC context, problems of drastically different size must be solved. Parallel solvers may not yield linear speedup and the efficiency may deteriorate on a large parallel computer system when the problems become too small. In this case, too little work can be executed concurrently and the scalar overhead dominates. This effect is well-known and can be understood prototypically in the form of Amdahl's law [21]. In general, we characterize a solver, when applied to a problem of given size, by its processor range for which the parallel efficiency remains above an acceptable threshold. For this purpose, we introduce the new notion of *scalability window*. Because of memory constraints, the scalability window will open at a certain minimal processor number. For larger processor numbers the parallel efficiency will deteriorate until the scalability window closes. In practice, additional restrictions imposed by the system and the software permit only specific processor numbers within the scalability window to be used. On the coarser MLMC levels, the problem size is in general too small to use the full machine. The problem is outside the scalability window and solver-parallelism alone is insufficient. On the finer levels we may not have enough samples to fill the entire machine, and thus sample-parallelism alone is insufficient. Especially for adaptive MLMC, where the number of samples on each level is not known a priori but must be

increased adaptively using data from all levels. Obtaining a highly efficient parallel algorithm on peta-scale systems is a challenging load balancing problem. Finding the optimal schedule restricted by a complex combination of mathematical and technical constraints is a high-dimensional, multi-constrained, discrete optimisation problem. Developing suitable approaches in this setting is one of our main objectives. See [32, 33] for earlier static and dynamic load balancing approaches, and [28] for adaptive multilevel stochastic algorithms.

The paper is structured as follows: In Section 2, we briefly review the MLMC method and its adaptive version. Section 3 introduces the model problem. Here, we use an alternative PDE-based sampling technique for Matérn covariances [24] that allows us to reuse the parallel multigrid solver. In Sections 4 and 5, we define a classification of different parallel execution strategies and develop them into different parallel scheduling approaches. In Section 6, we study the parallel efficiency, flexibility and robustness of the proposed strategies. Finally Section 7 reports on large-scale experiments on advanced supercomputer systems.

2. The Multilevel Monte Carlo method. To describe the MLMC method, we assume that we have a hierarchy of finite element (FE) discretisations of (1). Let $\{V_\ell\}_{\ell\geq 0}$ be a nested sequence of FE spaces with $V_\ell \subset V$, mesh size $h_\ell > 0$ and M_ℓ degrees of freedom. In the Hierarchical Hybrid Grids (HHG) framework [3, 18], the underlying sequence of FE meshes is obtained via uniform mesh refinement from a coarsest grid $\mathcal{T}_0$, and thus $h_\ell \simeq 2^{-\ell} h_0$ and $M_\ell \simeq 2^{3\ell} M_0$ in three space dimensions.

Denoting by $u_\ell = u_\ell(x,\omega) \in V_\ell$ the FE approximation of u on Level ℓ, we have

$$\mathcal{M}_\ell(u_\ell;\omega) = 0, \qquad \ell \geq 0. \tag{2}$$

2.1. Standard Monte-Carlo Simulation. The standard Monte Carlo (MC) estimator for the expected value $\mathbb{E}[Q]$ of $Q(u)$ on level $L \geq 0$ is given by

$$\widehat{Q}_L^{\mathrm{MC},N} = \frac{1}{N}\sum_{i=1}^{N} Q_L^i\,, \tag{3}$$

where $Q_L^i = Q_L(u_L^i,\omega^i)$, $i = 1,\ldots,N$, are N independent samples of $Q_L(u_L)$.

There are two sources of error: (i) The *sampling error* due to the finite number N of samples in (3), and (ii) the *bias error* due to the FE approximation. For the bias error, assuming that $|Q_L^i - Q(u^i,\omega^i)| = \mathcal{O}(M_L^{-\alpha})$, for almost all ω^i and a constant $\alpha > 0$, it follows directly that there exists a constant C_b, independent of M_L, such that $|\mathbb{E}[Q_L - Q]| \leq C_b M_L^{-\alpha}$ (cf. [34]).

The total error is typically quantified via the *mean square error* (MSE), given by

$$e\left(\widehat{Q}_L^{\mathrm{MC},N}\right)^2 := \mathbb{E}[(\widehat{Q}_L^{\mathrm{MC},N} - \mathbb{E}[Q])^2] = (\mathbb{E}[Q_L - Q])^2 + N^{-1}\mathbb{V}[Q_L], \tag{4}$$

where $\mathbb{V}[Q_L]$ denotes the variance of the random variable $Q_L(u_L)$. The first term in (4) can be bounded by ε_b^2 if $M_L \geq (\varepsilon_b/C_b)^{1/\alpha}$, and the second term is smaller than a sample tolerance ε_s^2 if $N \geq \mathbb{V}[Q_L]\varepsilon_s^{-2}$. We note that for L sufficiently large, $\mathbb{V}[Q_L] \approx \mathbb{V}[Q]$. To ensure that the total MSE is less than ε^2 we choose $\varepsilon_s^2 = \zeta\varepsilon^2$ and $\varepsilon_b^2 = (1-\zeta)\varepsilon^2$, for any fixed $0 < \zeta < 1$.

Thus, to reduce (4) we need to choose a sufficiently fine FE mesh and a sufficiently large number of samples. The cost for one sample Q_L^i of Q_L depends on the complexity of the FE solver and of the random field generator. Typically it will grow like $C_c M_L^\gamma$,

for some $\gamma \geq 1$ and some constant C_c, independent of i and of M_L. Thus, the total cost to achieve a MSE $e(\widehat{Q}_L^{\mathrm{MC},N})^2 \leq \varepsilon^2$ (the ε-*cost*) is

$$\mathrm{Cost}\left(\widehat{Q}_L^{\mathrm{MC},N}\right) = \mathcal{O}(M^\gamma N) = \mathcal{O}(\varepsilon^{-2-\gamma/\alpha}). \tag{5}$$

For the coefficient field and for the output functional studied below, we have only $\alpha = 1/6$ [34]. In that case, even if $\gamma = 1$, to reduce the error by a factor 2 the cost grows by a factor of $2^8 = 256$, which quickly leads to an intractable problem even in a massively parallel environment.

2.2. Multilevel Monte-Carlo Simulation. Multilevel Monte Carlo (MLMC) simulation [15, 6, 2] seeks to reduce the variance of the estimator by a hierarchy of FE models as control variates. Using the simple identity $\mathbb{E}[Q_L] = \mathbb{E}[Q_0] + \sum_{\ell=1}^{L} \mathbb{E}[Y_\ell]$, we estimate the mean on the coarsest level (Level 0) and correct this mean successively by adding estimates of the expected values of $Y_\ell(\omega) := Q_\ell(u_\ell, \omega) - Q_{\ell-1}(u_{\ell-1}, \omega)$, for $\ell \geq 1$, $Y_0 := Q_0$. The MLMC estimator is then defined as

$$\widehat{Q}_L^{\mathrm{ML}} := \sum_{\ell=0}^{L} \widehat{Y}_\ell^{\mathrm{MC},N_\ell}, \tag{6}$$

where the numbers of samples N_ℓ, $\ell = 0, \ldots, L$, are chosen to minimize the total cost for a given tolerance (see Eqn. (9) below). Note that we require the FE solutions $u_\ell(x, \omega^i)$ and $u_{\ell-1}(x, \omega^i)$ on two levels to compute a sample Y_ℓ^i of Y_ℓ, for $\ell \geq 1$, and thus two PDE solves, but crucially both with the same ω^i (see Algorithm 1).

Algorithm 1 Multilevel Monte Carlo.

for all levels $\ell = 0, \ldots, L$ **do**
  for $i = 1, \ldots, N_\ell$ **do**
    Set up (2) for ω^i on Level ℓ and $\ell - 1$ (if $\ell > 0$).
    Compute $u_\ell(\omega^i)$ and $u_{\ell-1}(\omega^i)$ (if $\ell > 0$), as well as Y_ℓ^i.
  end for
  Compute $\widehat{Y}_\ell^{\mathrm{MC},N_\ell} = \frac{1}{N_\ell} \sum_{i=1}^{N_\ell} Y_\ell^i$.
end for
Compute $\widehat{Q}_L^{\mathrm{ML}}$ using (6).

The cost of this estimator is

$$\mathrm{Cost}(\widehat{Q}_L^{\mathrm{ML}}) = \sum_{\ell=0}^{L} N_\ell \mathcal{C}_\ell, \tag{7}$$

where $\mathcal{C}_\ell$ is the cost to compute one sample of Y_ℓ on level ℓ. For simplicity, we use independent samples across all levels, so that the $L + 1$ standard MC estimators in (6) are independent. Then, the MSE of $\widehat{Q}_L^{\mathrm{ML}}$ simply expands to

$$e\left(\widehat{Q}_L^{\mathrm{ML}}\right)^2 = \left(\mathbb{E}[Q_L - Q]\right)^2 + \sum_{l=0}^{L} N_\ell^{-1} \mathbb{V}[Y_\ell]. \tag{8}$$

This leads to a hugely reduced variance of the estimator since both FE approximations Q_ℓ and $Q_{\ell-1}$ converge to Q and thus $\mathbb{V}[Y_\ell] \to 0$, as $M_{\ell-1} \to \infty$.

The bias error can be ensured to be less than ε_b by choosing $M_L \geq (\varepsilon_b/C_b)^{-1/\alpha}$ again, but there is some flexibility in choosing N_ℓ on each level to ensure the sampling error is less than ε_s^2. We can use this freedom to minimize the cost $\mathrm{Cost}(\widehat{Q}_L^{\mathrm{ML}})$ in (7) subject to the constraint $\sum_{\ell=0}^{L} N_\ell^{-1}\mathbb{V}[Y_\ell] = \varepsilon_s^2$, a simple discrete, constrained optimization problem with respect to $N_0, \ldots, N_L$ (cf. [15, 6]). It leads to

$$N_\ell \;=\; \varepsilon_s^{-2}\left(\sum_{\ell=0}^{L}\sqrt{\mathbb{V}[Y_\ell]\mathcal{C}_\ell}\right)\sqrt{\frac{\mathbb{V}[Y_\ell]}{C_\ell}}. \tag{9}$$

Finally, under the assumptions that $\mathcal{C}_\ell \leq C_c M_\ell^{\gamma}$ and $\mathbb{V}[Y_\ell] \leq C_v M_\ell^{-\beta}$ for some $0 < \beta \leq 2\alpha$ and $\gamma \geq 1$ and for two constants C_c and C_v, independent of M_ℓ, the ε-cost to achieve $e(\widehat{Q}_L^{\mathrm{ML}})^2 \leq \varepsilon^2$ can be bounded by

$$\mathrm{Cost}(\widehat{Q}_L^{\mathrm{ML}}) \;=\; \varepsilon_s^{-2}\left(\sum_{\ell=0}^{L}\sqrt{\mathbb{V}[Y_\ell]\mathcal{C}_\ell}\right)^2 \;\leq\; C_{\mathrm{ML}}\,\varepsilon^{-2-\max\left(0,\frac{\gamma-\beta}{\alpha}\right)}. \tag{10}$$

Typically $\beta \approx 2\alpha$ for smooth functionals $Q(\cdot)$. For CDFs, i.e., cumulative distribution functions, we typically have $\beta = \alpha$.

There are three regimes: $\gamma < \beta$, $\gamma = \beta$ and $\gamma > \beta$. In the case of the exponential covariance, typically $\gamma > \beta$ and $\beta = 2\alpha$ and thus $\mathrm{Cost}(\widehat{Q}_L^{\mathrm{ML}}) = \mathcal{O}(\varepsilon^{-\gamma/\alpha})$, which is a full two orders of magnitude faster than the standard MC method. Moreover, MLMC is *optimal* for this problem, in the sense that its cost is asymptotically of the same order as the cost of computing a single sample to the same tolerance ε.

2.3. Adaptive Multilevel Monte Carlo. In Algorithm 2 we present a simple sequential, adaptive algorithm from [15, 6] that uses the computed samples to estimate bias and sampling error and thus chooses the optimal values for L and N_ℓ. Alternative adaptive algorithms are described in [16, 7, 12]. For the remainder of the paper we will restrict to uniform mesh refinement, i.e. $h_\ell = 2^{-\ell}h_0$ and $M_\ell = \mathcal{O}(8^\ell M_0)$ in 3D.

Algorithm 2 Adaptive Multilevel Monte Carlo.

1: Set ε, ζ, $L = 1$ and $N_0 = N_1 = N_{\mathrm{Init}}$.
2: **for** all levels $\ell = 0, \ldots, L$ **do**
3: Compute new samples of Y_ℓ until there are N_ℓ.
4: Compute $\widehat{Y}_\ell^{\mathrm{MC},N_\ell}$ and s_ℓ^2, and estimate $\mathcal{C}_\ell$.
5: **end for**
6: Update the estimates for N_ℓ using (12).
7: **if** $\widehat{Y}_L^{\mathrm{MC},N_L} > (8^\alpha - 1)\varepsilon_b$ **then** increase $L \to L+1$ and set $N_L = N_{\mathrm{Init}}$
8: **if** all N_ℓ and L are unchanged **then go to** 9 **else go to** 2
9: Set $\widehat{Q}_L^{\mathrm{ML}} = \sum_{\ell=0}^{L}\widehat{Y}_\ell^{\mathrm{MC},N_\ell}$.

To estimate the bias error, let us assume that M_ℓ is sufficiently large, so that we are in the asymptotic regime, i.e. $|\mathbb{E}[Q_\ell - Q]| \approx C_b\, M_\ell^{-\alpha}$. Then (cf. [12])

$$|\mathbb{E}[Q_\ell - Q]| \leq \frac{1}{8^\alpha - 1}\widehat{Y}_\ell^{\mathrm{MC},N_\ell}. \tag{11}$$

Also, using the sample estimator $s_\ell^2 := \frac{1}{N_\ell}\sum_{i=1}^{N_\ell}\left(Y_\ell^i - \widehat{Y}_\ell^{\mathrm{MC},N_\ell}\right)^2$ to estimate $\mathbb{V}[Y_\ell]$

and the CPU times from the runs up-to-date to estimate $\mathcal{C}_\ell$, we can estimate

$$N_\ell \;\approx\; \varepsilon_s^{-2} \left(\sum_{\ell=0}^{L} \sqrt{s_\ell^2 \mathcal{C}_\ell} \right) \sqrt{\frac{s_\ell^2}{C_\ell}}. \tag{12}$$

3. Model problem and deterministic solver. As an example, we consider an elliptic PDE in weak form: Find $u(\cdot,\omega) \in V := H_0^1(D)$ such that

$$\int_D \nabla v(x) \cdot \big(k(x,\omega)\nabla u(x,\omega)\big)\,\mathrm{d}x = \int_D f(x)v(x)\,\mathrm{d}x, \quad \text{for all } v \in V \text{ and } \omega \in \Omega. \tag{13}$$

This problem is motivated from subsurface flow. The solution u and the coefficient k are random fields on $D \times \Omega$ related to fluid pressure and rock permeability. For simplicity, we only consider $D = (0,1)^3$, homogeneous Dirichlet conditions and a deterministic source term f. If $k(\cdot,\omega)$ is continuous (as a function of x) and $k_{\min}(\omega) := \min_{x\in\overline{D}} k(x,\omega) > 0$ almost surely (a.s.) in $\omega \in \Omega$, then it follows from the Lax-Milgram Lemma that this problem has a unique solution (cf. [4]). As quantities of interest in Section 7, we consider $Q(u) := u(x^*)$, for some $x^* \in D$, or alternatively $Q(u) := \frac{1}{|\Gamma|}\int_\Gamma -k\frac{\partial u}{\partial n}\,\mathrm{d}s$, for some two-dimensional manifold $\Gamma \subset \overline{D}$.

3.1. Discretisation. To discretise (13), for each $\omega \in \Omega$, we use standard $\mathbb{P}_1$ finite elements on a sequence of uniformly refined simplicial meshes $\{\mathcal{T}_\ell\}_{\ell\geq 0}$. Let V_ℓ be the FE space associated with $\mathcal{T}_\ell$, $\mathcal{N}_\ell$ the set of interior vertices, h_ℓ the mesh size and $M_\ell = |\mathcal{N}_\ell|$ the number of degrees of freedom. Now, problem (13) is discretised by restricting it to functions $u_\ell, v_\ell \in V_\ell$. Using the nodal basis $\{\phi_j : x_j \in \mathcal{N}_\ell\}$ of V_ℓ and expanding $u_\ell(\cdot,\omega) := \sum_{j\in\mathcal{N}_\ell} U_j^{(\ell)}(\omega)\phi_j$, this can be written as a linear equation system where the entries of the system matrix are assembled elementwise based on on a four node quadrature formula $A_{i,j}^{(\ell)}(\omega) := \sum_{\tau\in\mathcal{T}_\ell} \nabla\phi_i \cdot \nabla\phi_j\big|_\tau \frac{|\tau|}{4}\sum_{k=1}^4 k(x_k^\tau,\omega)$. Here x_k^τ, $1 \leq k \leq 4$ denote the four vertices of the element τ.

The quantity of interest $Q(u)$ is approximated by $Q(u_\ell)$. For $Q(u_\ell)$ to converge to $Q(u)$, as $\ell \to \infty$, we need stronger assumptions on the random field k. Let $k(\cdot,\omega) \in C^{0,t}(\overline{D})$, i.e. Hölder-continuous with coefficient $t \in (0,1)$, and suppose $k_{\min}(\omega)$ and $\|k(\cdot,\omega)\|_{C^{0,t}}$ have bounded second moments. It was shown in [34] that $\alpha = \frac{t}{3}$, which also implies the bound for the variance with $\beta = 2\alpha = \frac{2t}{3}$, since

$$\mathbb{V}\left[Q(u_\ell) - Q(u_{\ell-1})\right] \leq \mathbb{E}\left[(Q(u_\ell) - Q(u_{\ell-1}))^2\right] \leq 2\sum_{r=\ell,\ell-1} \mathbb{E}\left[(Q(u) - Q(u_r))^2\right].$$

3.2. PDE-based sampling for lognormal random fields. A coefficient function k of particular interest in subsurface flow applications [8, 4] is the lognormal random field $k(\cdot,\omega) := \exp(Z(\cdot,\omega))$, where $Z(\cdot,\omega)$ is a mean-free, stationary Gaussian random field with exponential covariance

$$\mathbb{E}[Z(x,\omega)Z(y,\omega)] = \sigma^2 \exp\left(-\frac{|x-y|}{\lambda}\right). \tag{14}$$

The two parameters in this model are the *variance* σ^2 and the *correlation length* λ. Individual samples $k(\cdot,\omega)$ of this random field are in $C^{0,t}(\mathbb{R}^3)$, for any $t < 1/2$. In particular, this means that the relevant convergence rates are $\alpha = 1/3 - \delta$ and $\beta = 2/3 - \delta$, for any $\delta > 0$. The field $Z(\cdot,\omega)$ belongs to the larger class of Matérn covariances [24, 25], which also includes smoother, stationary fields, but we will only consider the exponential covariance in this paper.

Two of the most common approaches to realise the random field Z above are Karhunen-Loeve (KL) expansion [14] and circulant embedding [9, 20]. While the KL expansion is very convenient for analysis and essential for polynomial expansion methods such as stochastic collocation, it can quickly dominate all the computational cost for short correlation lengths λ in three dimensions. Circulant embedding, on the other hand, relies on the Fast Fourier Transform, which may pose limits to scalability in a massively parallel environment. An alternative way to sample $Z(x,\omega)$ is to exploit the fact that in three dimensions, mean-free Gaussian fields with exponential covariance are solutions to the stochastic partial differential equation (SPDE)

$$(\kappa^2 - \Delta)Z(x,\omega) =^d W(x,\omega), \tag{15}$$

where the right hand side W is Gaussian white noise with unit variance and $=^d$ denotes equality in distribution. As shown by Whittle [39], a solution of this SPDE will be Gaussian with exponential covariance $\sigma^2 = (8\pi\kappa)^{-1}$ and $\lambda = 2/\kappa$.

In [24], the authors show how this SPDE can be solved using a FE discretisation and this will be the approach we use to bring our fast parallel multigrid methods to bear again. Since we only require samples of $k(\cdot,\omega) = \exp(Z(\cdot,\omega))$ at the vertices of $\mathcal{T}_\ell$, we discretise (15) using again standard $\mathbb{P}_1$ finite elements. If now $Z'_\ell(\cdot,\omega) \in V'_\ell$ denotes the FE approximation to Z', then we approximate $k(x_j,\omega)$ in our quadrature formula by $\exp(Z'_\ell(x_j,\omega))$, for all $x_j \in \mathcal{N}_\ell$. It was shown in [24, 29] that Z'_ℓ converges in a certain weak sense to Z' with $\mathcal{O}(M_\ell^{1/3-\delta})$, for any $\delta > 0$. Since (15) is in principle posed on all of $\mathbb{R}^3$ we embed the problem into the larger domain $\widetilde{D} := (-1,2)^3 \supset D$ with artificial, homogeneous Neumann boundary conditions on $\partial\widetilde{D}$ (see [29]).

4. Performance parameters and execution strategies. Although MLMC methods can achieve better computational complexity than standard MC methods, parallel execution strategies depend strongly on the performance characteristics of the solver. The ultimate goal is to schedule the different subtasks on the $P_{\max}$ processors such that the total run time is minimal. This can be formulated as a high dimensional, multi-constraint discrete optimization problem. More precisely, it is in general NP-complete, see, e.g., [11, 13, 22, 35] and the references therein.

4.1. Characteristic performance parameters. To design an efficient scheduling strategy, we rely on predictions of the time-to solution and have to take into account fluctuations. Static strategies thus possibly suffer from a significant load imbalance and may result in poor parallel efficiency. Dynamic strategies, such as the greedy load balancing algorithms in [32], which take into account run-time data are more robust, especially when run-times vary strongly within a level.

For the best performance, the number of processors P_ℓ per sample on level ℓ should lie within the scalability window $\{P_\ell^{\min}, P_\ell^{\min}+1, \ldots, P_\ell^{\max}\}$ of the PDE solver, where the parallel efficiency is above a prescribed threshold of, e.g., 80%. Due to machine constraints, P_ℓ may be restricted to a subset, such as $\{P_\ell^{\min}, 2P_\ell^{\min}, \ldots, 2^S P_\ell^{\min}\}$, where $S \in \mathbb{N}_0$ characterizes the size of the scalability window and $P_\ell^{\max} = 2^S P_\ell^{\min}$. Efficient modern implementations of 3D multigrid schemes have excellent strong scalability and a large scalability window, with typical values of $S = 4$ for a parallel efficiency threshold of 80%. In addition, the HHG solver also exhibits excellent weak scalability. We can thus assume that $P_\ell^{\min} = 2^{3\ell} P_0^{\min}$ and $P_\ell^{\max} = 2^{3\ell} P_0^{\max}$ for PDEs in 3D. The value of $P_0^{\min}$ is the number of processors for which the main memory capacity is fully utilized. Multigrid PDE solvers typically achieve the best parallel efficiency for $P = P_\ell^{\min}$, when each subdomain is as large as possible, and the ratio

of computation to communication is maximal (cf. [19]).

In the following, the time-to solution for the ith sample on level ℓ executing on $2^\theta P_\ell^{\min} = 2^{3\ell+\theta} P_0^{\min}$ processors is denoted by $t(i,\ell,\theta)$. We assume that

$$t(i,\ell,\theta) \approx C_{\ell,\theta}(\omega^i)\, t_{\ell,\theta}, \qquad 1 \le i \le N_\ell,\ 0 \le \ell \le L,\ 0 \le \theta \le S. \tag{16}$$

Here, $t_{\ell,\theta}$ is a reference time-to solution per sample on level ℓ, such as the mode, median, mean or minimum over a sample set, while $C_{\ell,\theta}(\omega^i)$ encapsulates fluctuations across samples. It depends on the robustness of the PDE solver, as well as on the type of parallel computer system. It is scaled to be one in the case of no run-time variations. Fig. 1 (right) shows a typical run-time distribution for 2048 samples each of which was computed on 512 processors with $\ell = 0$ and $\sigma^2 = 0.5$ in (14).

Assuming no efficiency loss due to load imbalances and an optimal parallel efficiency for $\theta = 0$, the theoretical optimal mean run-time for the MLMC method is

$$t_{\text{mlmc}}^{\text{opt}} = \frac{P_0^{\min}}{P_{\max}} \sum_{\ell=0}^{L} N_\ell 2^{3\ell} \mathbb{E}(C_{\ell,0}) t_{\ell,0}\,. \tag{17}$$

There are three main sources of inefficiency in parallel MLMC algorithms: (i) a partly idle machine due to large run-time variations between samples scheduled in parallel, (ii) non-optimal strong scalability properties of the solver, i.e., $t_{\ell,\theta} > 2t_{\ell,\theta-1}$, or (iii) over-sampling, i.e., more samples than required are scheduled to fill the machine. In the following we address (ii) and (iii) in more detail.

The strong scaling of a solver can be characterized by $\text{Eff}_\ell(\theta) := t_{\ell,0}/(2^\theta t_{\ell,\theta})$. In order to predict $t_{\ell,\theta}$, $1 \le \theta \le S$, we define a surrogate cost function depending on $0 \le \theta \le S$ that is motivated by Amdahl's law [21]:

$$t_{\ell,\theta} \approx t_{\ell,0}(B + 2^{-\theta}(1-B)), \quad \text{Eff}_\ell(\theta) \approx (2^\theta B + (1-B))^{-1}. \tag{18}$$

The serial fraction parameter B in (18) quantifies the amount of non-parallelizable work. It can be calibrated from time measurements. Fig. 1 (left) shows the HHG parallel efficiency for $S = 4$, $\ell = 0$ and $P_0^{\min} = 512$ on the JUQUEEN supercomputer (cf. Sec. 7) and the influence of different serial fraction parameters $B \in \{0, 0.01, 0.1, 1\}$. The fitted serial fraction parameter B lies in the range of $[0.01, 0.03]$ for different types of PDE within the HHG framework, and its parallel efficiency is in excellent agreement with Amdahl's law. Moreover B is almost constant over the levels so that we use a single value. In an adaptive strategy, we can also use performance measurements from past computations to reevaluate B for future scheduling steps.

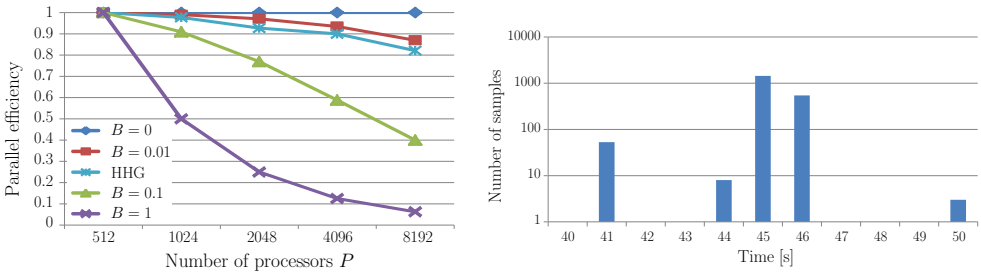

FIG. 1. *Left: Parallel efficiency for different serial fraction parameters B, Right: Example of a run-time histogram for a multigrid solver using full multigrid-cycles.*

Let $J_\ell(\theta) \in \mathbb{N}$ denote the maximum number of samples that can be computed simultaneously on level ℓ if $2^{3\ell+\theta} P_0^{\min}$ processors are used per sample, and let $k_\ell^{\text{seq}}(\theta)$ denote the number of required sequential cycles to run a minimum of N_ℓ samples. Then, $J_\ell(\theta)$, $k_\ell^{\text{seq}}(\theta)$ and the associated relative load imbalance $\text{Imb}_\ell(\theta)$ are given by

$$J_\ell(\theta) = \left\lfloor \frac{P_{\max}}{2^{3\ell+\theta} P_0^{\min}} \right\rfloor, \quad k_\ell^{\text{seq}}(\theta) = \left\lceil \frac{N_\ell}{J_\ell(\theta)} \right\rceil, \quad \text{Imb}_\ell(\theta) := 1 - \frac{2^{3\ell+\theta} P_0^{\min} N_\ell}{k_\ell^{\text{seq}}(\theta) P_{\max}}. \tag{19}$$

Note that $0 \leq \text{Imb}_\ell(\theta) < 1$, with $\text{Imb}_\ell(\theta) = 0$ for a perfectly balanced load and $\text{Imb}_\ell(\theta) > 0$ when part of the machine is idle due to $P_{\max}/(2^{3\ell+\theta} P_0^{\min}) \notin \mathbb{N}$ or $N_\ell / J_\ell(\theta) \notin \mathbb{N}$. The idle processors can be used to compute additional samples in the last sequential steps that are not necessary to achieve the required tolerance, but still improve the accuracy, or we can schedule samples on other levels in parallel (cf. Sect. 4.2). The product

$$\eta_\ell(\theta) := (1 - \text{Imb}_\ell(\theta)) \text{Eff}_\ell(\theta) \tag{20}$$

will be termed *MLMC level efficiency* and we note that it also depends on N_ℓ.

4.2. Classification of concurrent execution strategies. We classify execution strategies for MLMC methods in two ways, either referring to the layers of parallelisms or to the resulting time-processor diagram.

4.2.1. Layers of parallel execution. To fully exploit modern parallel systems we must identify multiple layers of parallelism. For MLMC, three natural layers exist:
Level parallelism: The estimators on level $\ell = 0, \ldots, L$ may be computed in parallel.
Sample parallelism: The samples $\{Y_\ell^i\}_{i=1}^{N_\ell}$ on level ℓ may be evaluated in parallel.
Solver parallelism: The PDE solver to compute sample Y_ℓ^i may be parallelized.

The loops over the levels and over the samples are inherently parallel, except for some minimal postprocessing to compute the statistical quantities of interest. The challenge is how to balance the load between different levels of parallelism and how to schedule the solvers for each sample. Especially in the adaptive setting, without a priori information, an exclusive use of level parallelism is not always possible, but in most practical cases, a minimal number of required levels and samples is known a priori. For the moment, we assume L and N_ℓ, $0 \leq \ell \leq L$, to be fixed and given. In general, these quantities have to be determined dynamically (cf. Alg. 2).

The concurrent execution can now be classified by the number of layers of parallelism that are exploited. Typically, $P_{\max} > \sum_{l=0}^{L} N_\ell$ and $P_{\max} \gg N_L$ on modern supercomputers. Thus, solver parallelism is mandatory for large-scale computing and the only possible one-layer approach is the solver-only strategy. For a two-layer approach, one can either add the level layer or the sample layer. Since the number of levels L is, in general, quite small, the solver-level strategy has significantly lower parallelization potential than the solver-sample strategy. Finally, the most flexible three-layer approach takes into account all three possible layers of parallelism.

4.2.2. Concurrency in the processor-time diagram. An alternative way to classify different parallel execution models is to consider the time-processor diagram, as illustrated in Fig. 2, where the scheduling of each sample Y_ℓ^i, $1 \leq i \leq N_\ell$, $0 \leq \ell \leq L$, is represented by a rectangular box with the height expressing the number of processors used. A parallel execution model is called *homogeneous bulk synchronous* if at any time in the processor diagram, all tasks execute on the same level with the same number of processors. Otherwise it is called *heterogeneous bulk synchronous*.

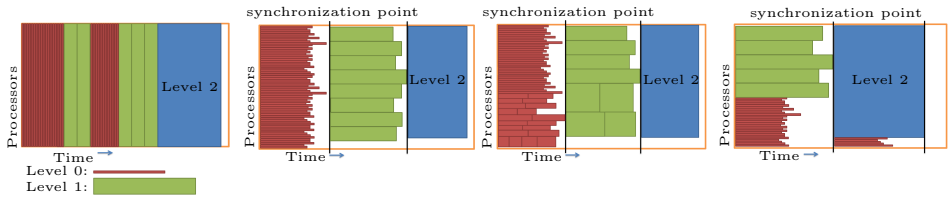

FIG. 2. *From left to right: illustration of homogeneous (one-layer and two-layer) and of heterogeneous bulk synchronous strategies (two-layer and three-layer).*

The one-layer homogeneous strategy, as shown in Fig. 2 (left), offers no flexibility. The theoretical run-time is simply given by $\sum_{\ell=0}^{L}\sum_{i=1}^{N_\ell} t(i,\ell,\theta_l^{\max})$, where $\theta_\ell^{\max}$ is such that $P_{\max} = 2^{3\ell+\theta_\ell^{\max}} P_0^{\min}$. It guarantees perfect load balancing, but will not lead to a good overall efficiency since on the coarser levels $\theta_\ell^{\max}$ is typically significantly larger than S. On the coarsest level we may even have $M_0 < P_{\max}$, i.e., less grid points than processors. Thus we will not further consider this option.

5. Examples for scheduling strategies. Our focus is on scheduling algorithms that are flexible with respect to the scalability window of the PDE solver and robust on huge numbers of processors $P_{\max}$. To solve the optimization problems, we will either impose additional assumptions that allow an exact solution, or we will use meta-heuristic search algorithms, such as simulated annealing [36, 37]. Before we start let us briefly comment on two important practical aspects for the implementation.

Sub-communicators. To parallelize over samples as well as within samples, we split the `MPI_COMM_WORLD` communicator via the `MPI_Comm_split` command and provide each sample with its own MPI sub-communicator. This requires only minimal changes to the multigrid algorithm and all MPI communication routines can still be used. A similar approach, using the MPI group concept, is used in [33].

Random number generator. To generate the samples of the diffusion coefficient $k(x,\omega)$ we use the approach described in Sect. 3.2. This requires suitable random numbers for the definition of the white noise on the right hand side of (15). For large scale MLMC computations we select the *Ran* [27] generator that has a period of $\approx 3.1\cdot 10^{57}$ and is thus suitable even for 10^{12} realizations. It is parallelized straightforwardly by choosing different seeds for each process, see, e.g., [23].

5.1. Sample synchronous and level synchronous homogeneous. Here we assume that the run-time of the solver depends on the level ℓ and on the number of associated processors, but not on the particular sample Y_ℓ^i, $1 \le i \le N_\ell$. As the different levels are treated sequentially and each concurrent sample is executed with the same number of processors, we can test all possible configurations.

Let $0 \le \ell \le L$ be fixed. Then, for a fixed $0 \le \theta \le S$, the total time on level ℓ is $k_\ell^{\text{seq}}(\theta)t_{\ell,\theta}$. We select the largest index $\theta_\ell \in \{0,1,\dots,S\}$ such that

$$\theta_\ell = \arg\min_{0\le\theta\le S} k_\ell^{\text{seq}}(\theta)t_{\ell,\theta} = \arg\max_{0\le\theta\le S} \text{Eff}_\ell(\theta)(1-\text{Imb}_\ell(\theta)) = \arg\max_{0\le\theta\le S} \eta_\ell(\theta), \tag{21}$$

i.e., minimizing run-time per level or equivalently, maximizing total level efficiency. Since $k_\ell^{\text{seq}}(\theta)$ is given explicitly in (19), the computation of θ_ℓ is trivial provided $t_{\ell,\theta}$ is known for all θ, e.g., using the average of pre-computed timings of the solver on level ℓ. Alternatively, we can use (18) with a fitted serial fraction B, such that

$$\theta_\ell = \arg\min_{0\le\theta\le S} k_\ell^{\text{seq}}(\theta)(B + 2^{-\theta}(1-B)).$$

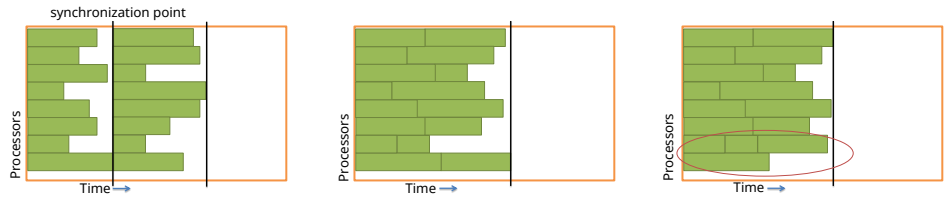

FIG. 3. *Illustration of different homogeneous scheduling strategies. Left: sample synchronous homogeneous (SaSyHom); Centre: level synchronous homogeneous (LeSyHom); Right: dynamic level synchronous homogeneous (DyLeSyHom, Sec. 5.3).*

Given θ_ℓ, we then group the processors accordingly and run $k_\ell^{\text{seq}}(\theta_\ell)$ sequential steps for each level ℓ. The actual value of $t_{\ell,0}$ does not affect the choice of θ_ℓ, but only the absolute run-time.

We consider two variants: (i) *Sample synchronous homogeneous (SaSyHom)* where a synchronization step is imposed after each sequential step (see Fig. 3, left). Here statistical quantities can be updated after each step. (ii) *Level synchronous homogeneous (LeSyHom)*, where each block of $2^{3\ell+\theta}P_0^{\min}$ processors executes all $k_\ell^{\text{seq}}(\theta_\ell)$ samples without any synchronization (see Fig. 3, centre). Altogether $k_\ell^{\text{seq}}(\theta_\ell)J_\ell(\theta_\ell) \geq N_\ell$ samples are computed. When the run-time does not vary across samples, both strategies will results in the same MLMC run-time. If it does vary then the LeSyHom strategy has the advantage that fluctuations in the run-time $t(i,\ell,\theta)$ will be averaged and a shorter overall MLMC run-time can be expected for sufficiently large $k_\ell^{\text{seq}}(\theta_\ell)$.

5.2. Run-time robust homogeneous. So far, we have assumed that the run-time is sample independent, which is idealistic. In the experiment in Fig. 1 (right), 3 out of 2048 samples required a run-time of 50 s on $P_0^{\min} = 512$ processors. On a large machine with $P_{\max} = 524288$ and with $\theta_0 = 0$ we need only $k_0^{\text{seq}}(0) = 2$ sequential steps on level 0. Therefore, the (empirical) probability that the SaSyHom strategy leads to a runtime of 100 s is about 75%, while the theoretical optimal run-time is $\frac{2}{2048}\sum_{i=1}^{2048} t_i \approx 90$ s. Here, t_i is the actual run-time of the ith sample from Fig. 1 (right). The probability that the LeSyHom strategy leads to a runtime of 100 s is less than 1%; in all other cases, a run-time of ≤ 96 s is achieved.

Let us now fix $0 \leq \ell \leq L$ again and include run-time variations in the determination of θ_ℓ. Unfortunately, in general, run-time distribution functions are not known analytically, and thus the expected run-time

$$E_{\ell,\theta} := \mathbb{E}\left[\max_{1\leq j\leq J_\ell(\theta)}\left(\sum_{k=1}^{k_\ell^{\text{seq}}(\theta)} t(i_{jk},\ell,\theta)\right)\right] \tag{22}$$

cannot be computed explicitly. Here, the samples are denoted by i_{jk} with $j = 1,\ldots,J_\ell(\theta)$ and $k = 1,\ldots,k_\ell^{\text{seq}}(\theta)$, related to their position in the time-processor diagram. The expression in (22) yields the actual, expected run-time on level ℓ to compute $J_\ell(\theta)k_\ell^{\text{seq}}(\theta) \geq N_\ell$ samples with $2^{3\ell+\theta}P_0^{\min}$ processors per sample when no synchronization after the sequential steps is performed.

The main idea is now to compute an approximation $\widehat{E}_{\ell,\theta}$ for $E_{\ell,\theta}$, and then to find θ_ℓ such that $\widehat{E}_{\ell,\theta_\ell} \leq \widehat{E}_{\ell,\theta}$, for all $0 \leq \theta \leq S$. First, we approximate $t(i_{jk},\ell,\theta)$ by $C_{\ell,\theta}(\omega^{i_{jk}})\,t_{\ell,\theta}$ in (16) and assume that $C_{\ell,\theta}(\cdot) \equiv C_{0,0}(\cdot)$, independent of ℓ and θ.

Approximating the expected value by an average over μ samples then leads to

$$\widehat{E}_{\ell,\theta}(\mu) := \frac{1}{\mu} \sum_{m=1}^{\mu} \max_{1 \le j \le J_\ell(\theta)} \left(\sum_{k=1}^{k_\ell^{\mathrm{seq}}(\theta)} C_{0,0}(\omega^{i_{jkm}}) \right) t_{\ell,\theta} \,, \tag{23}$$

where $i_{jkm} := i_{jk} + (m-1)J_\ell(\theta)k_\ell^{\mathrm{seq}}(\theta)$. If reliable data for $t_{\ell,\theta}$ is available we define

$$\theta_\ell := \arg \min_{0 \le \theta \le S} \widehat{E}_{\ell,\theta}(\mu)$$

Otherwise, we include, as before, a further approximation and replace $t_{\ell,\theta}$ by $B + 2^{-\theta}(1-B)$ in (23) before finding the minimum of $\widehat{E}_{\ell,\theta}(\mu)$. To decide on the number of samples μ in (23), we keep increasing μ until μ and $\mu/2$ yield the same θ_ℓ. For all our test settings, a value of $\mu \le 500$ was sufficient.

To evaluate (23), we need some information on the stochastic cost factor $C_{0,0}(\omega)$ which was assumed to be constant across levels and across the scaling window. We use a run-time histogram associated with level $\ell = 0$. This information is either available from past computations or can be built up adaptively within a MLMC method. Having the run-times t^k, $1 \le k \le K$, of K samples on level $\ell = 0$ at hand, we emulate $C_{0,0}(\omega)$ by using a pseudo random integer generator from a uniform discrete distribution ranging from one to K and replace the obtained value $j \le K$ by $t^j K / \sum_{k=1}^K t^k$. Having computed the value of θ_ℓ, we proceed as for LeSyHom and call this strategy *run-time robust homogeneous (RuRoHom)*. For constant run-times, RuRoHom yields again the same run-times as LeSyHom and as SaSyHom.

5.3. Dynamic variants. The use of pre-computed values for θ_ℓ and $k_\ell^{\mathrm{seq}}(\theta_\ell)$ in all the variants above will still lead to unnecessary inefficiencies in the case of large run-time variations. Instead, new samples can also be assigned to each processor block dynamically at run-time, as soon as a block terminates a computation, until the required number N_ℓ is reached. This can reduce the total run-time on level ℓ further, largely avoiding over-sampling. However, on massively parallel architectures this will only be effective if it does not lead to a significant communication overhead. The dynamic strategy can be combined with either the LeSyHom or the RuRoHom approach and we denote them *dynamic level synchronous homogeneous (DyLeSyHom)* and *dynamic run-time robust homogeneous (DyRuRoHom)*, respectively. Fig. 3 (right) illustrates the DyLeSyHom strategy. Here it is essential that not all processor blocks execute the same number of sequential steps.

To utilize the full machine, it is crucial that processors are not blocked by actively waiting to coordinate the asynchronous execution. The necessary functionality may not be fully supported on all parallel systems. We use the MPI 2.0 standard that permits one-sided communication and allows a non-intrusive implementation. The one-sided communication is achieved by remote direct memory access (RDMA) using registered memory windows. In our implementation, we create a window on one processor to synchronize the number of samples that are already computed. Exclusive locks are performed on a get/accumulate combination to access the number of samples.

5.4. Heterogeneous bulk synchronous scheduling. Heterogeneous strategies are clearly more flexible than homogeneous ones, but the number of scheduling possibilities grows exponentially. Thus, we must first reduce the complexity of the scheduling problem. In particular, we ignore again run-time variations and assume $t(i,\ell,\theta) = t_{\ell,\theta}$. We also assume that $N_\ell > 0$ on all levels $\ell = 0, \ldots, L$. Within an

adaptive strategy, samples may only be required on some of the levels at certain times and thus this condition has to hold true only on a subset of $\mathcal{I} := \{0, \dots, L\}$.

In contrast to the homogeneous setups, we do not aim to find scaling parameters θ_ℓ that minimize the run-time on each level separately, but instead minimize the total MLMC run-time. We formulate the minimization process as two nested constrained minimization steps. Let $N_{\ell,\theta} \in \mathbb{N}_0$ be the number of samples on level ℓ that are carried out in parallel with $2^{3\ell+\theta} P_0^{\min}$ processors. Then, firstly assuming $N_{\ell,\theta}$ to be given, for all $0 \leq \ell \leq L$, $0 \leq \theta \leq S$, we solve the following constrained minimization problem for $k_\ell^{\text{seq}}(\theta)$, the associated sequential steps,

$$\arg \min_{k_\ell^{\text{seq}}(\theta) \in \mathbb{N}_0} \left(\max_{0 \leq \theta \leq S} t_{\ell,\theta}\, k_\ell^{\text{seq}}(\theta) \right), \qquad \sum_{\theta=0}^{S} N_{\ell,\theta}\, k_\ell^{\text{seq}}(\theta) \geq N_\ell.$$

Secondly, having found $k_\ell^{\text{seq}}(\theta)$, we seek values for $N_{\ell,\theta} \in \mathbb{N}_0$ that minimize

$$\arg \min_{N_{\ell,\theta} \in \mathbb{N}_0} \max_{\substack{0 \leq \theta \leq S \\ 0 \leq \ell \leq L}} t_{\ell,\theta} k_\ell^{\text{seq}}(\theta),$$

the expected run-time, subject to the following inequality constraints

$$\sum_{\ell=0}^{L} \sum_{\theta=0}^{S} N_{\ell,\theta} 2^{3\ell} 2^{\theta} P_0^{\min} \leq P_{\max}, \tag{24a}$$

$$\sum_{\theta=0}^{S} N_{\ell,\theta} > 0, \text{ for } \ell \in \mathcal{I}. \tag{24b}$$

We apply integer encoding [30] for the initialization and for possible mutations and require that $N_{\ell,\theta} \in [0, 2^{-3\ell} 2^{-\theta} P_{\max}/P_0^{\min}]$. Condition (24a) is a hard constraint. The number of processors that are scheduled cannot be larger than $P_{\max}$. If (24a) is violated, we enforce it by a repeated multiplication of $N_{\ell,S}, \dots, N_{\ell,0}$ by $1/2$ until it holds. At first glance this might lead to a load imbalance, but the applied meta-heuristic search strategy compensates for it. Condition (24b) that at least one sample is scheduled on each level at all times could clearly be relaxed. However, this would require a redistribution of processors in the optimization problem and can significantly increase the algorithmic complexity, especially for large scale architectures. If (24b) is violated on some level ℓ, we set $N_{\ell,0} = 1$. With the values of $N_{\ell,\theta}$ identified, the samples are distributed dynamically onto the machine, see also [32].

For the following two subsections, let us consider the example $(N_0, N_1, N_2, N_3) = (4123, 688, 108, 16)$ and actual run-times from a set of numerical experiments:

$$(t_{\ell,\theta})_{\substack{0 \leq \ell \leq 3, \\ 0 \leq \theta \leq 4}} = \begin{pmatrix} 167 & 83.84 & 42.30 & 21.63 & 11.60 \\ 171 & 86.28 & 44.53 & 23.13 & 12.41 \\ 177 & 90.40 & 47.07 & 24.21 & 12.97 \\ 179 & 91.61 & 48.27 & 24.86 & 13.63 \end{pmatrix}. \tag{25}$$

5.4.1. The degenerate case $S = 0$ and a new auxiliary objective. A cheap but non-optimal way to choose $N_{\ell,0}$ for $S = 0$, is

$$N_{\ell,0} = \left\lfloor \frac{P_{\max}\, N_\ell\, t_{\ell,0}}{\sum_{i=0}^{L} N_i 2^{3i} P_0^{\min} t_{i,0}} \right\rfloor. \tag{26}$$

The corresponding run-time is then $\max_{\ell=0,\dots,L} t_{\ell,0}\lceil N_\ell/N_{\ell,0}\rceil$, and the total number of processors is $\sum_{\ell=0}^{L} N_{\ell,0}2^{3\ell}P_0^{\min}$. This choice is acceptable when the workload is evenly distributed across levels and serves as a good initial guess in combination with an adaptive local neighborhood search.

The first column of (25) gives a total run-time of 716 s in (26) and the distribution $(N_{0,0}, N_{1,0}, N_{2,0}, N_{3,0}) = (1314, 221, 36, 5)$, see the left of Fig. 4, while on the right, a pattern that leads to the minimal run-time of 684 s is illustrated. This configuration to achieve the minimal run-time is not unique. Due to weak scaling effects, the lower levels tend to have a larger number of sequential steps than the higher ones.

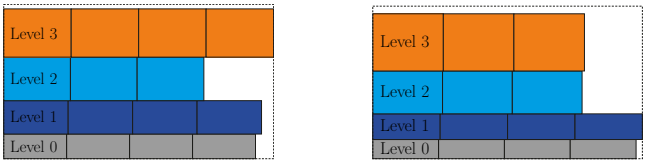

FIG. 4. *Different scheduling patterns: Selection of $N_{\ell,0}$ by (26) (left) and optimal choice of $N_{\ell,0}$*

For $S > 0$, we cannot define a good starting guess as easily and have to resort to meta-heuristic strategies. We consider simulated annealing (SA), see, e.g., [36, 38], which provides a computationally feasible approach to approximately solve complex scheduling problems. We start with $S = 0$. The following experiments were performed with Python using *inspyred*[1] with minor modifications. The *temperature* in the SA method is decreased using a geometric schedule $T_{k+1} = 0.8\,T_k$ with initial temperature $T_0 = 10^3$, which is of the order of the initial changes of the objective function.

We choose a Gaussian mutation with distribution $\mathcal{N}(0, 0.1\,P_{\max}/(2^{3\ell}P_0^{\min}))$ and mutation rate 0.2, guaranteeing that roughly one gene per SA iteration is changed. All runs were repeated ten times with different seeds, and we report minimal (min), maximal (max) and mean (avg) MLMC run-times. Selecting a stopping criterion of 1 000 evaluations in SA, we obtain $[\min, \text{avg}, \max] = [684, 691.2, 708]$ s, while after 2 000 evaluations we obtain $[\min, \text{avg}, \max] = [684, 684, 684]$ s. The curve labelled "time [w/o aux. obj.]" in Fig. 5 shows the evolution of the average MLMC run-time between iteration 100 and 1 000 in the SA. We observe that between iteration 250 and iteration 800 almost no decrease in the average run-time is achieved. This is due to the rather flat structure of the objective function in large parts of the search domain since many different possible combinations yield identical run-times.

To improve the performance of the SA scheduling optimizer, we introduce the number of idle processors as a second auxiliary objective. Having more idle processors, the probability to find a shorter run-time in the next search step is increased and thus the total number of iterations is reduced significantly, see Fig. 5. A MLMC run-time of less than 700 s can now be found in less than 300 iterations. The optimal run-time can be obtained with $(N_{0,0}, N_{1,0}, N_{2,0}, N_{3,0}) = (1031, 172, 36, 6)$ and a total of 7783 processors used. From now on, we always include the number of idle processors as a secondary objective in the SA optimization algorithm.

5.4.2. The highly scalable case $S = 4$ and new hybrid mutants. We use the data in (25) and compare different mutation operators. Standard strategies [26], such as random reset mutation and non-uniform mutation, do not improve efficiency. Since more than 50 000 SA iterations are necessary to find average run-times close to

[1] Garrett, A. (2012). inspyred (Ver. 1.0). Inspired Intelligence Initiative; from http://github.com

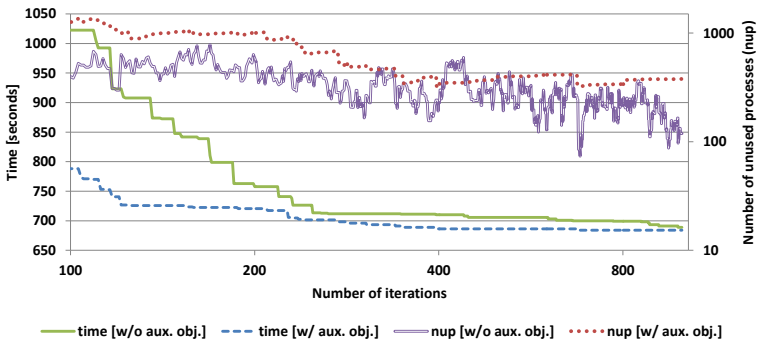

FIG. 5. *Average MLMC run-time with $P_{max} = 8192$ and number of unused processors (nup) with and without auxiliary objective (w/ and w/o aux. obj.) w.r.t. the number of SA iterations.*

the optimal one, new problem-adapted mutation operators are essential.We propose two new hybrid variants. Both first perform a Gaussian mutation with mutation rate 0.1 and then take the required processor numbers into account.

Hybrid A. In each step, we select randomly two different "genes" N_{ℓ_1,θ_1} and N_{ℓ_2,θ_2}, $0 \le \ell_1, \ell_2 \le L$, $0 \le \theta_1, \theta_2 \le S$, as well as a uniformly distributed random number $k \in \{0, .., N_{\ell_1,\theta_1} - 1\}$. Then we mutate

$$N_{\ell_1,\theta_1} = N_{\ell_1,\theta_1} - k \quad \text{and} \quad N_{\ell_2,\theta_2} = N_{\ell_2,\theta_2} + \left\lfloor k\, 2^{\theta_1-\theta_2} 2^{3(\ell_1-\ell_2)} \right\rfloor .$$

If the original values N_{ℓ_1,θ_1} and N_{ℓ_2,θ_2} are admissible (satisfy the constraints (24)), then the mutated genes are also admissible. This type of mutation exploits level parallelism and the scalability window of the solver. For $S = 0$, it reduces to exploiting the weak scalability of the solver to balance the workload on each level separately.

Hybrid B. This variant is most suitable for PDE solvers with large scalability windows. It is identical to Hybrid A except for keeping $\ell_1 = \ell_2$ fixed, therefore exploiting only solver parallelism and not level parallelism.

TABLE 1

Comparison of MLMC run-times (min, avg, max) using measurements from JUQUEEN (cf. Sec. 7) for different mutation operators and different numbers of SA iterations.

Mutation	1 000	4 000	16 000	64 000
Gaussian	612.0, 633.9, 641.2	605.2, 624.8, 635.5	603.9, 614.6, 624.6	603.9, 608.0, 612.0
Hybrid A	624.6, 632.7, 641.2	603.9, 608.0, 612.0	604.5, 604.5, 604.5	604.5, 604.5, 604.5
Hybrid B	603.9, 619.8, 627.3	603.9, 603.9, 603.9	603.9, 603.9, 603.9	603.9, 603.9, 603.9

In Tab. 1, we see that Hybrid B performs best. It is less sensitive to the initial guess than Hybrid A and robustly finds a very efficient scheduling in less than 4 000 SA iterations. Thus, we restrict ourselves to Hybrid B type mutations in the following examples. In the example considered in this section, it leads to the schedule

$$(27)\quad (N_{\ell,\theta})_{\substack{0\le\ell\le 3,\\ 0\le\theta\le 4}} = \begin{pmatrix} 0 & 443 & 73 & 0 & 0 \\ 1 & 98 & 0 & 0 & 0 \\ 0 & 0 & 3 & 3 & 0 \\ 6 & 0 & 0 & 0 & 0 \end{pmatrix} \quad (k^{\text{seq}}_{\ell,\theta})_{\substack{0\le\ell\le 3,\\ 0\le\theta\le 4}} = \begin{pmatrix} 0 & 7 & 14 & 0 & 0 \\ 3 & 7 & 0 & 0 & 0 \\ 0 & 0 & 12 & 24 & 0 \\ 3 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Comparing the results for $S = 0$ and $S = 4$, shows how important the strong scaling of the solver is to reach shorter MLMC run-times. It allows to reduce the run-time

by more than 10%, and thus improves the parallel MLMC performance significantly. If strong scaling is included ($S > 0$), we call this scheduling strategy *StScHet* in the following. Otherwise, if no strong scaling is included ($S = 0$), we call the scheduling strategy *noStScHet*.

Finally, we summarise all the considered scheduling strategies in Tab. 2.

TABLE 2
Summary of parallel scheduling strategies.

Abbreviation	Schedule	Defined in
SaSyHom	Sample Synchronous Homogeneous	Sec. 5.1
LeSyHom	Level Synchronous Homogeneous	Sec. 5.1
RuRoHom	Run-Time Robust Homogeneous	Sec. 5.2
DyLeSyHom	Dynamic Level Synchronous Homogeneous	Sec. 5.3
DyRuRoHom	Dynamic Run-Time Robust Homogeneous	Sec. 5.3
StScHet	Heterogeneous with Strong-Scaling ($S > 0$)	Sec. 5.4
noStScHet	Heterogeneous without Strong-Scaling ($S = 0$)	Sec. 5.4

6. Scheduling comparison. In this section, we evaluate the sampling strategies from the previous section and illustrate the influence of the serial fraction parameter B, of the level-averaged number of sequential steps and of the run-time variation.

6.1. The influence of the number of sequential steps. The fact that processor and sample numbers have to be integer not only complicates the solution of the optimization problem, it also strongly influences the amount of imbalance.

Let us start with some preliminary considerations, ignoring for the moment run-time variations. Now, let $\Delta t \geq 0$ denote the relative difference between the run-time $\sum_{\ell=0}^{L} k_\ell^{\text{seq}}(\theta_\ell) t_{\ell,\theta_\ell}$ of the presented strategies and the theoretically optimal run-time in (17) (with $\mathbb{E}(C_{\ell,0}) = 1$). Recall the MLMC level efficiency $\eta_\ell(\theta_\ell)$ in (20). Then

$$\Delta t = \frac{P_{\max}}{P_0^{\min}} \frac{\sum_{\ell=0}^{L} k_\ell^{\text{seq}}(\theta_\ell) t_{\ell,\theta_\ell}}{\sum_{\ell=0}^{L} N_\ell 2^{3\ell} t_{\ell,0}} - 1 = \frac{\sum_{\ell=0}^{L} N_\ell 2^{3\ell} t_{\ell,0} (\eta_\ell(\theta_\ell))^{-1}}{\sum_{\ell=0}^{L} N_\ell 2^{3\ell} t_{\ell,0}} - 1 \ .$$

For the special case that $P_{\max}/(2^{3\ell+\theta} P_0^{\min}) \in \mathbb{N}$, we can further bound Δt in terms of $k_{\text{seq}} := \sum_{\ell=0}^{L} N_\ell 2^{3\ell} P_0^{\min}/P_{\max}$. We assume that $t_{0,0} \leq t_{\ell,0} \leq t_{L,0}$, for all $\ell = 0, \ldots, L$, which is typically the case. The ratio $t_{L,0}/t_{0,0}$ reflects the weak scalability of the solver. Peta-scale aware massively parallel codes have a factor close to one. Recall from (25) that for our solver $t_{L,0}/t_{0,0} = 179/167 \approx 1.07$. Since $k_\ell^{\text{seq}}(\theta_\ell) t_{\ell,\theta_\ell} \leq k_\ell^{\text{seq}}(0) t_{\ell,0}$ and since $k_\ell^{\text{seq}}(0) \leq N_\ell 2^{3\ell} \frac{P_0^{\min}}{P_{\max}} + 1$ we have

$$\Delta t \leq \left(\frac{\max_{0 \leq \ell \leq L} t_{\ell,0}}{\min_{0 \leq \ell \leq L} t_{\ell,0}} \right) \frac{P_{\max}}{P_0^{\min}} \frac{L+1}{\sum_{\ell=0}^{L} N_\ell 2^{3\ell}} \ = \ \frac{t_{L,0}}{t_{0,0}} \frac{L+1}{k_{\text{seq}}}.$$

The larger k_{seq}, the smaller the efficiency loss.

In Fig. 6, we illustrate the influence of k_{seq} on LeSyHom, noStScHet, StScHet. We use $P_0^{\min} = 1$, $P_{\max} = 131072$ and $(N_0, N_1, N_2, N_3) = \text{fac} \cdot (4123, 688, 108, 16)$ with $\text{fac} \in \{1, \ldots, 18\}$, yielding $k_{\text{seq}} \approx 0.189 \cdot \text{fac}$. All strategies stay below the theoretically predicted upper bound. The two scheduling strategies that exploit the solver parallelism, LeSyHom and StScHet, are significantly more robust with respect to k_{seq} than the heterogeneous strategy, noStScHet, which ignores the scalability

window and sets $S = 0$. This is particularly relevant for adaptive MLMC strategies, where N_ℓ may be increased within any of the adaptive steps and then a new optimal scheduling pattern has to be identified. As a consequence of the ceil operator we see a staircase pattern for noStScHet. Regardless, all three strategies seem to be very robust to variations in k_{seq} and very efficient. The run-times for LeSyHom and StScHet are larger than the optimal one by roughly a factor of 1.5 for $k_{\text{seq}} \approx 0.2$ and only by a factor of 1.15 for $k_{\text{seq}} \approx 3.4$.

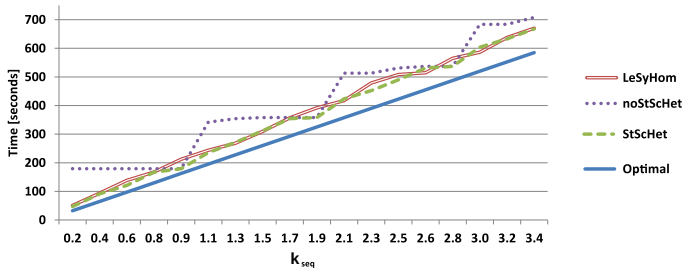

FIG. 6. *Heterogeneous versus homogeneous scheduling for* $k_{seq} \in [0.2, 3.4]$.

6.2. The influence of solver scalability. The serial fraction parameter B models the strong scalability of the solver, see Sec. 5. The higher B, the less beneficial it is to increase θ. In Fig. 7, we consider the influence of B on the run-time for two different values of fac, namely 4 and 16, and compare LeSyHom and StScHet using the same setup as in the previous subsection.

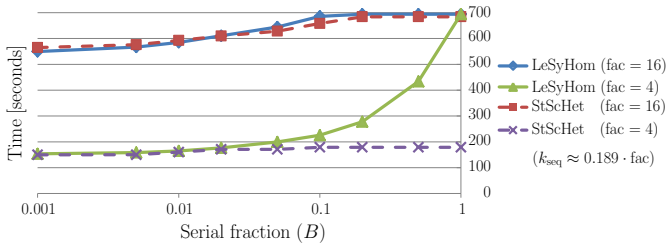

FIG. 7. *Influence of the serial fraction parameter on the run-time.*

First, we consider a small value $k_{\text{seq}} \approx 0.75$. With a serial fraction parameter $B \leq 0.02$, there is almost no run-time difference between the two strategies. For B up to 0.1 the run-time difference is below 25%. So the homogeneous strategy provides enough flexibility to be efficient in the case of large scalability windows with small values of B. However, the run-time increases significantly for LeSyHom for larger B, since the strong scaling of the solver is too poor to obtain a robust scheduling. Only a heterogeneous strategy with its flexibility to schedule parallel samples on different levels can guarantee small run-times in that case. The run-time of StScHet depends only moderately on the serial fraction parameter B.

The situation is different for larger values of k_{seq}. Then both strategies exhibit roughly the same performance, but the total run-time is more sensitive to the size of B. A good strong scaling of the PDE solver can improve the time to solution by up to 27% for the homogeneous and up to 21% for the heterogeneous bulk synchronous case.

As expected, carrying out one synchronization step with $k_{\text{seq}} \approx 3$ is more efficient than four steps with $k_{\text{seq}} \approx 0.75$. This observation is important for the design of efficient adaptive strategies, i.e., they should not be too fine granular. For highly performant multigrid solvers, i.e., $B \leq 0.05$, the simpler homogeneous strategies are an excellent choice. When the parallel scalability of the solver is poorer, which is typical for the peta-scale regime, i.e. near the strong scaling limit of the solver, the more complex heterogeneous strategies lead to significant efficiency gains.

6.3. Robustness and efficiency with respect to the parameters. In this subsection, we modify all three key parameters that we have discussed so far. We assume again that the run-time variations $C_{\ell,\theta}(\cdot)$ are independent of ℓ and θ and use a half-normal distribution with parameter Var for $C_{0,0}(\cdot) - 1$ follows a half-normal distribution, i.e. the mode of $C_{0,0}(\cdot)$ is at 1. The time $t_{\ell,\theta}$ is chosen to be the run-time of the mode, as described in Sec. 4.1.

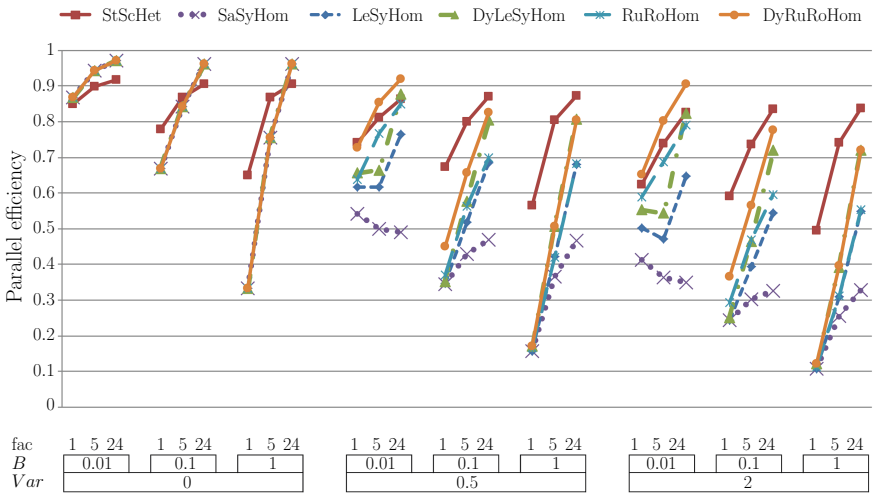

FIG. 8. *MLMC efficiencies for different values of the parameters $k_{seq} \approx 0.97 \cdot fac$, B and Var, and for all the different scheduling strategies.*

Fig. 8 illustrates the parallel efficiencies of all the strategies developed above (cf. Tab. 2), as well as their robustness with respect to the parameters k_{seq}, B and Var. The parallel efficiency is calculated with respect to the theoretical, optimal run-times given in (17) not with actual measured run-times. We choose $B \in \{0.01, 0.1, 1\}$ and $Var \in \{0, 0.5, 2\}$, and set the sample numbers on the different levels to be $(N_0, N_1, N_2, N_3) = \text{fac} \cdot (1314, 221, 36, 5)$, with $\text{fac} \in \{1, 5, 24\}$ and $P_{\max}/P_0^{\min} = 8192$.

We comment first on the case of no run-time variations, i.e., $Var = 0$, where our numerical results confirm that all homogeneous strategies produce the same performance. For large numbers of sequential steps, the homogeneous variants are superior to the heterogeneous ones. This is somewhat artificial and mainly due to the constraint (24b) which forces us to consider all levels in parallel. As mentioned above, this constraint is not essential and dropping it might lead to more efficient heterogeneous strategies. This will be the subject of future work. If variations in the run-time are included, then all homogeneous strategies yield different results. The parallel efficiency of the simplest one, SaSyHom, then drops to somewhere between 0.1 and 0.55. As expected, the worst performance is observed for a small k_{seq}, poor solver scalability, and

high run-time variation. In that case, the dynamic variants can counterbalance the run-time variations more readily and provide computationally inexpensive scheduling schemes (provided the technical realization is feasible).

Secondly, we discuss the case of small k_{seq}, typical for large parallel systems or when using adaptive MLMC. Here, only the heterogeneous strategies lead to acceptable parallel efficiencies for all values of B. The homogeneous variants result in efficiencies below 0.7 and 0.4, for $Var = 0$ and for $B = 0.1$ and $B = 1$, respectively.

In all considered cases, one of our strategies results in parallel efficiencies of more than 0.5; in many cases even more than 0.7. For moderate run-time variations and large enough k_{seq}, the parallel efficiency of StScHet improves to more than 0.8. StScHet is also the most robust strategy with respect to solver scalability. However, for solvers with good scalability, i.e. $B \leq 0.05$, the DyRoRuHom strategy is an attractive alternative, since it does not require any sophisticated meta-heuristic scheduling algorithm and can dynamically adapt to run-time variations in the samples.

7. Numerical results for MLMC. In this section, the scheduling strategies developed above are tested in a large-scale MLMC computation. We consider the model problem in (13) with $D = (0,1)^3$ and $f \equiv 1$, discretised by piecewise linear FEs. The serial fraction parameter for our multigrid PDE solver is $B \leq 0.02$ and the fluctuations in run-time are $< 2\%$. Only few timings deviate substantially from the average (cf. Fig. 1) so that we focus on investigating strategies for that regime.

The following experiments were carried out on the peta-scale supercomputer JUQUEEN, a 28 rack BlueGene/Q system located in Jülich, Germany[2]. Each of the 28 672 nodes has 16 GB main memory and 16 cores operating at a clock rate of 1.6 GHz. The compute nodes are connected via a five-dimensional torus network. HHG is compiled by the IBM XL C/C++ Blue Gene/Q, V12.0 compiler suite with MPICH2 that implements the MPI-2 standard and supports RDMA. Four hardware threads can be used on each core to hide latencies. We always use 2 processes (threads) per core to maximize the execution efficiency.

7.1. Static scheduling for scenarios with small run-time variations. We choose four MLMC levels, i.e., $L = 3$, with a fine grid that has roughly $1.1 \cdot 10^9$ mesh nodes. The random coefficient is assumed to be lognormal with exponential covariance, $\sigma^2 = 1$ and $\lambda = 0.02$. The quantity of interest is the PDE solution u evaluated at the point $x = (0.25, 0.25, 0.25)$. All samples are computed using a single FMG-2V(4,4) cycle on a fixed hierarchy of geometric levels, i.e., a full multigrid method with two V-cycles per new level, as well as four pre- and four post-smoothing steps. The excellent numerical and parallel efficiency of this multigrid method applied to (13) is well documented in [18]. In particular, after completing the FMG-2V(4,4) cycle for all samples, the minimal and maximal residual differ at most by a factor 1.5 within each MLMC level. We use an a priori strategy with $(N_\ell)_{\ell=0,1,2,3} = (4\,123, 688, 108, 16)$ for the MLMC estimator based on pre-computed variance estimates. We first study the balance between sample and solver parallelism and thus the tradeoffs between the efficiency of the parallel solver and possible load imbalances in the sampling strategy, as introduced in Sec. 4.1. We set $P_0^{\min} = 1$, and consequently $P_\ell^{\min} = 2^{3\ell}$. The run-times to compute a single sample with $P_\ell^{\min}$ processors are measured as $(t_{\ell,0})_{\ell=0,1,2,3} = (166, 168, 174, 177)$ seconds, showing only a moderate increase in run-time and confirming the excellent performance of the multigrid solver.

[2] http://www.fz-juelich.de/ias/jsc/EN/Expertise/Supercomputers/JUQUEEN/Configuration/Configuration_node.html

A lower bound for the run-time of the parallel MLMC estimator of $t^{\text{opt}}_{\text{mlmc}} = 520\,\text{s}$ is now provided by eq. (17). A static cost model is justified since the timings between individual samples vary little. We therefore employ the level synchronous homogeneous (LeSyHom) scheduling strategy, as introduced in Sect. 5.1. This requires only a few, cheap real time measurements to configure the MLMC scheduling strategy. The smaller θ, the larger the solver efficiency $\text{Eff}_\ell(\theta)$ while the larger θ, the smaller $\text{Imb}_\ell(\theta)$. To study this effect quantitatively, we measure the parallel solver efficiency in a pre-process step by computing a few samples on the coarsest level for different θ and measure the required time $t_{0,\theta}$. On level $\ell = 0$ we obtain $\{\text{Eff}_0(0), \text{Eff}_0(1), \text{Eff}_0(2), \text{Eff}_0(3), \text{Eff}_0(4)\} = \{1, 0.99, 0.96, 0.92, 0.86\}$ and by assuming that the efficiency is level independent, we take the same values on the higher levels.

In Tab. 3, we present for this particular N_ℓ the level efficiency $\eta_\ell(\theta)$ defined in (20), and the run-time as a function of θ. For $\theta = 0$ and $\ell = 0$, the runs are carried out with one thread, and they go up to 8 192 hardware threads on 4 096 cores for $\theta = 4$ and $\ell = 3$. The maximal level efficiency and the minimal run-time on each level are marked in boldface to highlight the best setting chosen by the LeSyHom strategy (cf. (21)).

TABLE 3
Level efficiencies, level run-times and total run-time in seconds

θ	time	$\eta_0(\theta)$	time	$\eta_1(\theta)$	time	$\eta_2(\theta)$	time	$\eta_3(\theta)$	t. time
0	167	0.50	171	0.67	177	0.84	**179**	**1.00**	694
1	168	0.50	173	0.67	181	0.84	183	0.99	704
2	127	0.64	**134**	**0.86**	188	0.81	193	0.96	642
3	108	0.74	139	0.83	**169**	**0.89**	199	0.92	**615**
4	**104**	**0.77**	136	0.84	181	0.81	218	0.86	640

The minimal run-time of a simple scheduling strategy that picks a fixed θ for all MLMC levels is 615 s. Our homogeneous scheduling strategies pick θ_ℓ for each level automatically and by doing so, a considerably shorter run-time of 586 s is obtained. If the solver scaled perfectly also to processor numbers that are not powers of 2, we could reduce the compute time even further by about 11% from 586 to 520 seconds. In summary, the strong scalability of the PDE solver helps to avoid load imbalances due to oversampling and improves the time to solution by about 15% from 694 to 586 seconds. The cost is well distributed across all levels, although most of the work is on the finest level which is typical for this model problem (cf. [4]).

We conclude this subsection with a strong scaling experiment, i.e., we increase the number of processes in order to reduce the overall time to solution. Since we are interested in the behavior for extremely large $P_{\max}$, we reduce the number of samples to $N_l = (1\,031, 172, 27, 4)$. The time for one MLMC computation with an increasing number of processes $P_{\max}$ is presented in Fig. 9. The initial computation employs $P_{\max} = 2\,048$ which is large enough so that all fine-grid samples can be computed concurrently. We scale the problem up to 131 072 processes. Increasing $P_{\max}$ while keeping the number of samples per level fixed results in a decrease of k_{seq}. Thus the load imbalance increases and the total parallel efficiency decreases. Nevertheless, even with $P_{\max} = 32768$, we obtain a parallel efficiency of over 60%. For $P_{\max} = 131072$, the efficiency drops below 40%, since we have reached the limits of the scalability window. This is inevitable in any strong scaling scenario. Overall, the compute time for the MLMC estimator can be reduced from 616 to 22 seconds, resulting in the

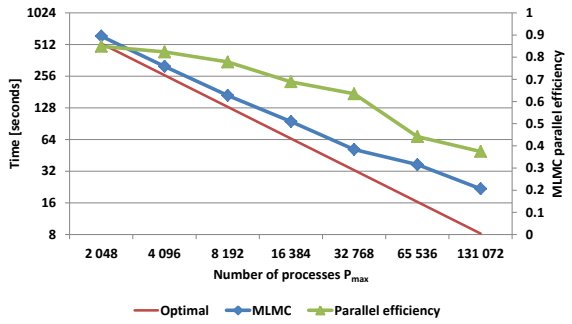

FIG. 9. *Strong scaling of MLMC using homogeneous bulk synchronous scheduling.*

claimed excellent parallel efficiency of the MLMC implementation. For each choice of $P_{\max}$, we select the optimal regime for θ_ℓ, $\ell = 0, \ldots, 3$, as discussed above.

7.2. Scheduling with dynamic variants for scenarios with large run-time variations. In this subsection, we compare static strategies with their dynamic variants in the case of large run-time variations on three MLMC levels, i.e. $L = 2$, with a fine grid resolution of about $1.6 \cdot 10^7$ mesh nodes. We assume again a lognormal random coefficient, but in order to increase the run-time variation, we choose a greater variance $\sigma^2 = 4.0$ and $\lambda = 0.5$. The quantity of interest, is the PDE solution u evaluated at the point $x = (0.5, 0.5, 0.5)$. All samples are computed with a FMG-2V(4,4) cycle with up to a hundred additional V(4,4) cycles until a relative residuum of 10^{-10}. Pre-computed variance estimates lead to an a priori strategy with $(N_\ell)_{\ell=0,1,2} = (27\,151, 10\,765, 3\,792)$. In this scenario, with $P_{\max} = 8192$, the estimated scaling parameters using the LeSyHom strategy are $(\theta_\ell^{\mathrm{LeSy}})_{\ell=0,1,2} = (1, 0, 0)$, whereas using the RuRoHom strategy yields $(\theta_\ell^{\mathrm{RuRo}})_{\ell=0,1,2} = (1, 1, 0)$. In Tab. 4, we compare the required time with both strategies in the static and dynamic variants and see that the dynamic strategies are 6% and 7% faster than their static counterparts.

TABLE 4
Comparison of static and dynamic scheduling strategies

Level	LeSyHom	DyLeSyHom	Ratio	RuRoHom	DyRuRoHom	Ratio
0	500 s	460 s	0.92	501 s	460 s	0.92
1	1512 s	1347 s	0.89	1449 s	1270 s	0.88
2	5885 s	5596 s	0.95	5885 s	5594 s	0.95
Total	**7897 s**	**7403 s**	**0.94**	**7835 s**	**7324 s**	**0.93**

7.3. Adaptive MLMC. Finally, we consider an adaptive MLMC algorithm as introduced in Sec. 2.3 in a weak scaling scenario with LeSyHom scheduling strategy.

Each row in Tab. 5 summarizes one adaptive MLMC computation. The MLMC method is initially executed on 2 048 cores, choosing $P_{\ell=0,1,2,3} = (2, 16, 128, 1\,024)$. In each successive row of the table, the number of unknowns on the finest level in the MLMC method and the number of processors on each level is increased by a factor of eight. Moreover, the correlation length λ of the coefficient field is reduced by a factor of two, keeping $\sigma^2 = 1$ fixed. This means that the problems are actually getting more difficult as well. The quantity of interest is defined as the flux across a separating

TABLE 5
Weak scaling of an adaptive MLMC estimator.

Processes	Resolution	Runtime	No. Samples Fine	No. Samples Total	Correlation length	Idle time
4 096	$1\,024^3$	$5.0 \cdot 10^3$ s	68	13 316	1.50E-02	3%
32 768	$2\,048^3$	$3.9 \cdot 10^3$ s	44	10 892	7.50E-03	4%
262 144	$4\,096^3$	$5.2 \cdot 10^3$ s	60	10 940	3.75E-03	5%

TABLE 6
Number of samples and over-samples for different levels for the largest run.

Level	No. partitions	No. Samples Scheduled	No. Samples Calculated	No. Over-samples Estimated	No. Over-samples Actual
0	2 048	7 506	8 192	3 726	686
1	256	2 111	2 304	429	193
2	32	382	384	15	2
3	4	57	60	3	3

plane Γ at $x_2 = 0.25$, i.e.,

$$Q(u, \omega) = \int_\Gamma k(x, \omega) \frac{\partial u}{\partial n} ds \,.$$

In all cases, the initial number of samples is set to $N_{\ell=0,1,2,3} = (1\,024, 256, 64, 16)$. The final number of samples is then chosen adaptively by the MLMC algorithm. It is listed in the table. As motivated in Sec. 2, the tolerance for the sampling error, which is needed in (12) to adaptively estimate N_ℓ, is chosen as $\varepsilon_s \approx |\mathbb{E}[Q_L - Q_{L-1}]|$, balancing the sampling error with the bias error. The estimates for the expected

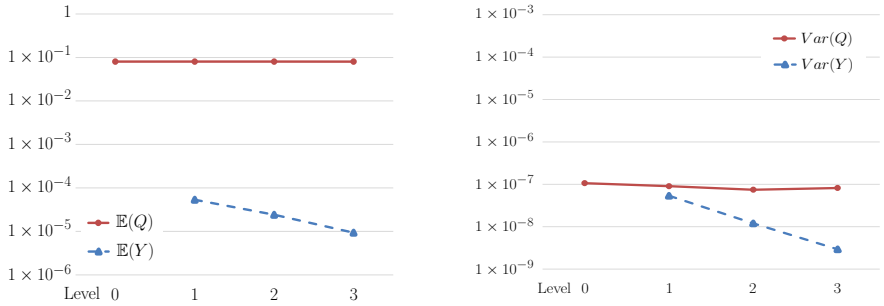

FIG. 10. *MLMC performance plots: expected value (left) and variance (right) of Q_ℓ (red, solid) and Y_ℓ (blue, dashed) for $\lambda = 0.015$ and $\sigma^2 = 1$.*

values and for the variances of Q_ℓ and Y_ℓ, for a problem of size $M_\ell = 1\,024^3$ and with a correlation length of $\lambda = 0.015$, are plotted in Fig. 10. The expected values and the variances of Y_ℓ show the expected asymptotic behavior as ℓ increases, confirming the benefits of the multilevel approach. The total number of samples that are computed is 13 316, but only 68 of them on the finest grid. A standard Monte Carlo estimator would require several thousand samples on level 3 and would be significantly more costly. The idle time, in the last column of Tab. 5, accounts for the variation in the number of V-cycles, required to achieve a residual reduction of 10^{-5} on each level within each call to the FMG algorithm.

The largest adaptive MLMC computation shown in Tab. 5 involves a finest grid with almost 7×10^{10} unknowns. Discrete systems of this size must be solved 60 times, together with more than 10 000 smaller problems, the smallest of which still has more than 1.6×10^7 unknowns. With the methods developed here, a computation of such magnitude requires a compute time of less than 1.5 hours when 131 072 cores running 262 144 processes are employed. Additional details for this largest MLMC computation are presented in Tab. 6. The table lists the number of partitions that are used on each level for the respective problem sizes. The number of calculated samples on each level is a multiple of the number of these partitions. As Tab. 6 illustrates, the number of scheduled samples is smaller and the difference indicates the amount of oversampling. The number of unnecessary samples is presented explicitly in the last column of the table to compare with the estimated number of unneeded samples. This estimated number is significantly higher on each level, since it is the sum of all the oversampled computations in all stages of the adaptive MLMC algorithm. As we pointed out earlier, this is caused by a special feature of the adaptive MLMC algorithm. Samples that were predicted to be redundant in an early stage of the algorithm, may become necessary later in the computation. Thus at termination, the actual oversampling is significantly less than predicted. This is a dynamic effect that cannot be quantified easily in a static a priori fashion.

8. Conclusions. In this paper we have explored the use of multilevel Monte Carlo methods combined with multigrid solvers on very large supercomputers. Three levels of parallelism must be coordinated, since it is not sufficient to just execute samples in parallel. The combination of solver- and sample-parallelism leads to a non-trivial scheduling problem, where the trade-off between solver scalability, oversampling, and additional efficiency losses due to run-time variations must be balanced with care. This motivated the development of scheduling strategies of increasing complexity, including advanced dynamic methods that rely on meta-heuristic search algorithms. These scheduling algorithms are based on performance predictions for the individual tasks that can in turn be derived from run-time measurements and performance models motivated by Amdahl's law.

The success of the techniques and their scalability are demonstrated on a large-scale model problem. The largest MLMC computation involves more than 10 000 samples and a fine grid resolution with almost 7×10^{10} unknowns and is thus one of the largest UQ computations demonstrated to date. It is executed on 131 072 cores of a peta-scale class supercomputer in 1.5 hours of total compute time.

The techniques of this paper may be extended by exploring other interesting dynamic load balancing strategies. In this case, the system is filled with jobs while prioritizing large jobs over small jobs. After a job terminates, the released processors can be used for the next job of same size or smaller jobs, if no large jobs are left. If new large jobs are created, then a sufficient number of smaller jobs must terminate to free enough space for the larger ones. The resulting gaps in processor utilization must be reduced as much as possible by suitable strategies. Heuristic approaches like this are of great interest when an even larger number of parallel processors becomes available. Then the complexity of the optimization problems, as they are solved here for the heterogeneous strategies, grows even worse. The dynamic variants of our homogeneous strategies have already showed improvements in run time and therefore energy consumption as compared to their static counterparts.

[1] A. H. Baker, R. D. Falgout, T. Gamblin, T. V. Kolev, M. Schulz, and U. M. Yang. Scaling algebraic multigrid solvers: On the road to exascale. In *Competence in High Performance Computing 2010*, pages 215–226. Springer, 2012.

[2] A. Barth, C. Schwab, and N. Zollinger. Multi-level Monte Carlo finite element method for elliptic PDE's with stochastic coefficients. *Numer. Math.*, 119:123–161, 2011.

[3] B. K. Bergen and F. Hülsemann. Hierarchical hybrid grids: data structures and core algorithms for multigrid. *Numer. Lin. Alg. Appl.*, 11(2-3):279–291, 2004.

[4] J. Charrier, R. Scheichl, and A. L. Teckentrup. Finite element error analysis of elliptic PDEs with random coefficients and its application to multilevel Monte Carlo methods. *SIAM J. Numer. Anal.*, 51(1):322–352, 2013.

[5] E. Chow, R. D. Falgout, J. J. Hu, R. S. Tuminaro, and U. M. Yang. A survey of parallelization techniques for multigrid solvers. In M. A. Heroux, P. Raghavan, and H. D. Simon, editors, *Parallel Processing for Scientific Computing*, chapter 10, pages 179–201. SIAM, 2006.

[6] K. A. Cliffe, M. B. Giles, R. Scheichl, and A. L. Teckentrup. Multilevel Monte Carlo methods and applications to elliptic PDEs with random coefficients. *Comput. Visual. Sci.*, 14(1):3–15, 2011.

[7] N. Collier, A. L. Haji-Ali, F. Nobile, E. von Schwerin, and R. Tempone. A continuation multilevel Monte Carlo algorithm. *BIT Numer. Math.*, 55(2):399–432, 2015.

[8] G. Dagan. *Flow and Transport in Porous Formations*. Springer, 1989.

[9] C. R. Dietrich and G. H. Newsam. Fast and exact simulation of stationary Gaussian processes through circulant embedding of the covariance matrix. *SIAM J. Sci. Comput.*, 18:1088–1107, 1997.

[10] J. Dongarra. Report on the Sunway TaihuLight system. Technical report, University of Tennessee, Oak Ridge National Laboratory, June 24, 2016. http://www.netlib.org/utk/people/JackDongarra/PAPERS/sunway-report-2016.pdf.

[11] M. Drozdowski. *Scheduling for Parallel Processing*. Springer-Verlag, London, 2009.

[12] D. Elfverson, F. Hellman, and A. Målqvist. A multilevel Monte Carlo method for computing failure probabilities. *SIAM/ASA J. Uncertainty Quantification*, 4(1):312–330, 2016.

[13] M. Garey and D. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co, New York, 1990.

[14] R. G. Ghanem and P. Spanos. *Stochastic Finite Elements: A Spectral Approach*. Springer, 1991.

[15] M. B. Giles. Multilevel Monte Carlo path simulation. *Operations Res.*, 56(3):981–986, 2008.

[16] M. B. Giles and B. J. Waterhouse. Multilevel quasi-Monte Carlo path simulation. *Radon Series Comp. Appl. Math.*, 8:1–18, 2009.

[17] B. Gmeiner, M. Huber, L. John, U. Rüde, and B. Wohlmuth. A quantitative performance study for Stokes solvers at the extreme scale. *Journal of Computational Science*, 17, Part 3:509 – 521, 2016. Recent Advances in Parallel Techniques for Scientific Computing.

[18] B. Gmeiner, U. Rüde, H. Stengel, C. Waluga, and B. Wohlmuth. Towards textbook efficiency for parallel multigrid. *Numer. Math. Theor. Meth. Appl.*, 8(01):22–46, 2015.

[19] T. Gradl, C. Freundl, H. Köstler, and U. Rüde. Scalable multigrid. In *High Performance Computing in Science and Engineering, Garching/Munich 2007*, pages 475–483. Springer, 2009.

[20] I. G. Graham, F. Y. Kuo, D. Nuyens, R. Scheichl, and I. H. Sloan. Quasi-Monte Carlo methods for elliptic PDEs with random coefficients and applications. *J. Comput. Phys.*, 230:3668–3694, 2011.

[21] G. Hager and G. Wellein. *Introduction to high performance computing for scientists and engineers*. CRC Press, 2010.

[22] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling. *Annals of Discrete Mathematics*, 1:343–362, 1977.

[23] J. L. Leva. A fast normal random number generator. *ACM Transactions on Mathematical Software (TOMS)*, 18(4):449–453, 1992.

[24] F. Lindgren, H. Rue, and J. Lindström. An explicit link between gaussian fields and gaussian markov random fields: the stochastic partial differential equation approach. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(4):423–498, 2011.

[25] G. J. Lord, C. Powell, and T. Shardlow. *An Introduction to Computational Stochastic PDEs*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2014.

[26] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 1996.

[27] W. H. Press. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, 2007.

[28] E. Prudencio and S. H. Cheung. Parallel adaptive multilevel sampling algorithms for the bayesian analysis of mathematical models. *International Journal for Uncertainty Quan-*

tification, 2(3), 2012.

[29] D. Simpson, J. Illian, F. Lindgren, S. Sørbye, and H. Rue. Going off grid: Computationally efficient inference for log-Gaussian Cox processes. *Biometrika*, 103:49–70, 2016.

[30] M. Srinivas and L. M. Patnaik. Genetic algorithms: A survey. *Computer*, 27(6):17–26, 1994.

[31] E. Strohmaier, H. W. Meuer, J. Dongarra, and H. D. Simon. The Top500 list and progress in high-performance computing. *Computer*, 48(11):42–49, 2015.

[32] J. Šukys. Adaptive load balancing for massively parallel multi-level Monte Carlo solvers. In *Parallel Processing and Applied Mathematics*, pages 47–56. Springer, 2014.

[33] J. Šukys, S. Mishra, and C. Schwab. Static load balancing for multi-level Monte Carlo finite volume solvers. In *Parallel Processing and Applied Maths*, pages 245–254. Springer, 2012.

[34] A. L. Teckentrup, R. Scheichl, M. B. Giles, and E. Ullmann. Further analysis of multilevel MC methods for elliptic PDEs with random coefficients. *Numer. Math.*, 125(3):569–600, 2013.

[35] J. D. Ullman. NP-complete scheduling problems. *J. Comput. System Sci.*, 10:384–393, 1975.

[36] P. J. M. Van Laarhoven and E. H. L. Aarts. *Simulated annealing*. Springer, 1987.

[37] P. J. M. Van Laarhoven, E. H. L. Aarts, and J. K. Lenstra. Job shop scheduling by simulated annealing. *Operations Res.*, 40(1):113–125, 1992.

[38] I. Wegener. Simulated annealing beats Metropolis in combinatorial optimization. In L. Caires et al., editors, *Automata, Languages and Programming*, volume 3850 of *Lecture Notes in Computer Science*, pages 589–601. Springer, 2005.

[39] P. Whittle. Stochastic processes in several dimensions. *Bull. Inst. Internat. Stat.*, 40:974–994, 1963.

[40] D. Xiu and G. E. Karniadakis. The Wiener–Askey polynomial chaos for stochastic differential equations. *SIAM J. Sci. Comput.*, 24(2):619–644, 2002.